AFRL-RH-WP-TP-2008-0007

Cognitive Design Patterns

Christopher R. Hale

Science Applications International Corporation
Dayton OH 45431

Vincent Schmidt

Air Force Research Laboratory
Cognitive Systems Branch

June 2008

Interim Report

Air Force Research Laboratory
Human Effectiveness Directorate
Warfighter Interface Division
Cognitive Systems Branch
Wright-Patterson AFB OH 45433-7022

NOTICE

AFRL-RH-WP-TP-2008-0007

//SIGNED//
VINCENT A. SCHMIDT
Work Unit Manager
Cognitive Systems Branch

//SIGNED//
DANIEL G. GODDARD
Chief, Warfighter Interface Division
Human Effectiveness Directorate
Air Force Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
June 2008	Conference Proceedings	

4.TITLE AND SUBTITLE

Cognitive Design Patterns

5a. CONTRACT NUMBER
FA8650-04-D-6405

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
62202F

6. AUTHOR(S)

[1]Christopher R. Hale, [2]Vincent Schmidt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
7184HEX6

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

[1]Science Applications International Corporation
Dayton OH 45431

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

[2]Air Force Materiel Command
Air Force Research Laboratory
Human Effectiveness Directorate
Warfighter Interface Division
Cognitive Systems Branch
Wright-Patterson AFB OH 45433-7022

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RHCS

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

AFRL-RH-WP-TP-2008-0007

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

WorldComp 2008, Las Vegas, NV, July 14-17, 2008.
88th ABW/PA cleared on 25 June, 2008, WPAFB-08-3825

14. ABSTRACT

We introduce the concept of cognitive design patterns and discuss ways in which these patterns can better integrate early work analyses with software development. Cognitive design patterns are units of work that, in combination, enable human operators to accomplish the range of tasks needed for success in complex systems. Each pattern consists of a normative model of the relevant cognitive competency, expressed in terms accessible to software design and practice. Our proposal is that these patterns be included as resources in GUI builders, thereby adding standardized design capabilities to the software engineering toolkit.

15. SUBJECT TERMS
Cognition, HCI, Design Patterns

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Vincent A. Schmidt
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR	8	**19b. TELEPHONE NUMBER** *(include area code)*

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. 239.18

THIS PAGE LEFT INTENTIONALLY BLANK

Cognitive Design Patterns

Christopher R. Hale
Science Applications International Corporation
Dayton, OH 45431

Vincent Schmidt
Air Force Research Laboratory
Wright-Patterson Air Force Base
Dayton, OH, USA

Abstract We introduce the concept of cognitive design patterns and discuss ways in which these patterns can better integrate early work analyses with software development. Cognitive design patterns are units of work that, in combination, enable human operators to accomplish the range of tasks needed for success in complex systems. Each pattern consists of a normative model of the relevant cognitive competency, expressed in terms accessible to software design and practice. Our proposal is that these patterns be included as resources in GUI builders, thereby adding standardized design capabilities to the software engineering toolkit.

Keywords: Cognition, HCI, Design Patterns

1.0 Introduction

One of the enduring frustrations of both the Cognitive Systems Engineering (CSE) and software development communities is the apparent inability to communicate with one another when designing complex systems. The CSE community has developed many methods and tools that can lead to a thorough understanding of the nature of work and the human-centered requirements for a system. However, this understanding often is not adequately conveyed to the development community in a way that is actionable. That is, there exists no good way to translate "requirements," as defined by the cognitive systems engineer, to code used by the software engineer to instantiate a working system. Fundamentally, this problem is comprised of two parts, both of which fall primarily at the feet of the cognitive system engineer. First, the content produced by the CSE community often is inadequate for the needs of the system and software engineering communities, often addressing issues that do not bear on design needs at a level of analysis that the engineering community can effectively use. Second, the information produced by the CSE community often is not communicated to the engineering communities in ways that enable incorporation of the information into ongoing design process. To be useful, CSE analysis results should be articulated in the language of requirements, specifications and normative principles. Instead, these analyses often are stated in the language of theories, controlled experiments and idiosyncratic results.

One solution to these problems would be to develop the kinds of content that lend themselves to requirements, specifications and normative principles and then to communicate this content to the engineering development process in the language of engineering development. An important concept used by the software engineering community to define and document the content of useful concepts is that of software design patterns. We are attempting to extend the design pattern concept to the problem of integration between CSE and engineering development by defining Cognitive Design Patterns (CDP). These are normative units of cognition that, when combined in different ways, can account for the work required in any context. Work elements upon which such patterns are based are identified and characterized during the CSE analysis that is carried out in the early stages of system design. Cognitive units are derived from these work elements and are "parameterized" for the specific design problem at hand. These normative cognitive units, along with other information obtained from the CSE analysis, are combined with other elements of system analysis to formulate system requirements and build a system model.

In addition to using the work elements to define and specify cognitive components at the design level, it is also possible to use the elements within a structured software development environment. Contemporary graphical user interfaces (GUIs) are being created by programmers using GUI-builder programs that specialize in GUI layout and corresponding code generation. Since cognitive work elements often model processes that include visual or procedural components, a list of work element "widgets" could be added as a GUI builder module. Such an implementation makes the cognitive component of design look even more like "cognitive design patterns" to both the designer and the implementer.

Adding a set of cognitive work element widgets to a GUI builder application brings two distinct advantages. First, relevant visualizations for the corresponding cognitive tasks could be suggested to the coder. This enables the programmer to more easily select appropriate techniques for implementing the cognitive specifications without fear of misrepresenting the requirements. Once a cognitive work element widget is chosen, a coding wizard might be used to assist developers with selecting and parameterizing an appropriate visualization, and a code template could

provide the basis for the resulting code in the programming language of choice.

Second, direct selection of the work element from a list of cognitive work element widgets promotes direct traceability back to the requirements. This level of traceability is useful for ensuring that all requirements have been accounted for in the system. Nearly as important, this traceability can be used to justify the reasons a system has a particular "look and feel," answering such questions as why a screen or system function looks or behaves a certain way.

The next section of this paper introduces the concept of cognitive work elements in more detail, providing an enumerated list of (what the authors would claim as) a comprehensive set of cognitive operations. These work elements are a valuable part of the initial system specification, providing early systems engineering inputs into the system design process.

For software systems, the paper also describes a mechanism by which developers can directly include these work elements in the application by selecting work elements as "widgets" within a GUI builder. This approach provides programmers a guided method for implementing cognitive requirements, and also shows end-to-end traceability from the requirements to the final software application.

Before proceeding to our own ideas and research, tt must be mentioned that others, especially within the human factors community, have also worked to define design patterns. Most of these works concentrate on high-level patterns of work and cognition, however, and fall short of introducing a mechanism through which a technologist has direct access to use the pattern in a software system implementation as we highlight below. See [1-4] for excellent examples of previous work.

2.0 Work Elements

From the point of view of Cognitive Systems Engineering (CSE), traceability is a crucial challenge to successful design. By traceability we mean the ability to relate originating (from the Cognitive Work Analysis (CWA)) and functional requirements defined early in a development program to the final artifacts that constitute the system under development. Ideally, one should be able to relate these requirements to the resulting artifacts through the design commitments made at the various stages along the development path of the system. Thus, when customers and users of the system ask "why does this artifact look the way it does and behave the way it does," the designer of a traceable system should be able to "reverse engineer" the artifact back through each series of design decisions to the original requirements contained in the CWA.

How does one explicitly relate the information contained in the CWA to the resulting artifact, and do so in terms of the engineering process that (should) form the bridge between these two points? Our approach has been to find ways to integrate the cognitive requirements with all stages of the system engineering process, thereby ensuring that the artifact that is eventually built is formed and constrained by this information. We do this by developing a matrix that explicitly relates the Cognitive Workflow Elements (CWE) identified in the original CWA to the system requirements resulting from early system analysis and modeling.

We define CWE as "units" of workflow required of the human system component to carry out elements of work that a system is being built to accomplish. The CWE for a particular system are identified by analyzing the contents of the CWA, critical decision analysis and other analyses carried out in the early stages of a development effort. Typically, a small set of CWE will result from this analysis. For example, consider a CWA for a visualization system designed to support operational assessment. Based on a set of concept maps developed through documentation, observation and detailed interviews with Subject Matter Experts (SME), we identified the CWE shown in Table 1.

Table 1. CWE for Operational Assessment

Acquire	Communicate	Compare	Infer
Decide	Discriminate	Estimate	Integrate
Assign	Aggregate	Evaluate	Identify
Choose	Describe	Generate	Interpret
Classify	Detect	Match	Plan
Monitor	Recognize	Prioritize	Verify

These elements are adequate to encompass all of the cognitive workflow required to carry out operational assessment. After the element set is identified, we develop conceptual definitions for each element. For example, our definition for *detect*, as carried out within the context of operational assessment, was: *Become aware of the existence of an object, value or attribute.* We then create a matrix that juxtaposes the CWE against the system requirements. A fragment of this matrix for an operational assessment visualization system is shown in Table 2, with system requirements in the left-most column and the CWEs from Table 1 across the top.

Notice that some cells of this matrix contain check marks. These indicate that the corresponding CWE participates in satisfying the system requirement in the

adjacent row. Thus, each row specifies the various combinations of CWE required for effective operation of the system, where effective operation is a function of adherence to the system requirements. This will explicitly link the CWE to the resulting system design through the system requirements, thereby enabling traceability. There is one further, and crucial, step to be taken once the CWE have been identified, conceptually defined and mapped to the system requirements. This is to develop models of the CWE.

Table 2. Traceability Matrix for an Operational Assessment Visualization System

System Requirement	Acquire	Aggregate	Assign	Choose	Classify	Communicate	Compare	Decide	Derive	Describe	Detect	Discriminate	Estimate	Evaluate	Generate	Infer	Integrate	Identify	Interpret	Match	Monitor	Plan	Prioritize	Recognize	Verify
The system shall aid in determining the potential effect of weather, terrain, and/or air spaces current/future on possible plan changes	√												√			√						√			
The system shall aid in determining if tactical objectives will be achieved	√				√								√			√			√					√	
The system shall aid in determining if operational objectives will be achieved	√				√								√			√			√					√	
The system shall allow and aid in the assessment of both direct and indirect effects of air, space and IO on the	√				√									√			√		√						
The system shall allow and aid in the derivement of the intended and unintended consequences of air ops wrt platforms, munitions, culture, population	√				√				√								√								
The system shall allow and aid in determining when and what to report to JFACC						√		√															√		
The system shall aid in determining the difference between the plan and actual					√																			√	
The system shall aid in determining how mission failures will affect the overall plan																√									
The system shall aid in determining how effective mission successes were													√												
The system shall aid in determining if we are behind plan														√											
The system shall aid in determining if we are ahead of schedule	√				√								√	√								√			

Development of such models makes the system requirements executable. With executable system requirements, expressed through rigorous models of the CWE, it is possible to create integrated system simulations that will enable tradeoff analyses to be carried out prior to specification of detailed software requirements or development of physical system concepts. To facilitate this process we develop normative models of each CWE that will allow exploration of the variables and parameters expected to affect system effectiveness.

For example, consider the following requirement for the operational assessment visualization system mentioned above:

The system shall provide a way to derive intended and unintended effects from tactical assessment results.

Referring to concept maps developed in the initial CWA, we find that this requirement can be satisfied with a combination of the following CWE: Inference, acquisition, classification, detection, evaluation, interpretation, and recognition. The executable model of this requirement will be comprised of submodels for each CWE, organized into a task network model of the overall work environment. We normally use a higher-level, discrete-event performance modeling package, such as the Combat Automation Requirements Testbed (CART) to carry out these simulations. When combined with environment and system models these human operator modeling packages enable us to express broad ranges of human performance within the context of overall systems, thereby allowing study of the kinds of constraints that will limit performance. Consider, for example, the interpretation element of the above requirement, where interpretation is defined as determining the task-relevant value of data or information. We assume that interpreted value is a function of the timeliness (T) and credibility (C) of the data or information received. Further assume that timeliness follows a sigmoid function in which the timeliness of information ranges from 1 immediately after it becomes available down to an asymptotic value approaching zero after many hours. In this case we can define interpreted value as:

$$V = TC$$

Where: $T = 1 - [1/1 + e^{-t}]$ and

$$C = \log_b(\Sigma z_i v_i)$$

Developing similar models for each CWE allows us to model ranges of performance for each of the requirements, with individual low-level models being connected together through the human system model. This allows us to model a wide range of work demands on the human component of the system and to produce estimates of human performance that can be used by other members of the system development team to conduct trade studies. These models also allow us to evaluate the conformance of human operators to requirements as the requirements are further refined to include system performance or effectiveness specifications, thereby connecting the humans to measurement of overall system effectiveness.

With executable models of the requirements in hand, the cognitive system engineers can then begin developing visualization concepts for system interfaces. Each cognitive work element has basic visualization requirements defined for it that we assume are consistent across contexts. For example, the cognitive element *compare* involves examining two or more objects in terms of their similarities and differences. The visualization requirements for this element include displaying to users the attributes and values of objects being compared. Further, the display should facilitate the comparison being carried out, for example, by highlighting the similarities and differences through some method of ranking, coding or some other means. As this example shows, there will be a basic structure associated with each element as well as performance parameters for the elements. Parameters for *compare* might include the number of to-be-compared elements that can be held in short-term memory and sensitivity limitations on attribute similarity used in comparisons.

The requirements provide the context, constraints and boundaries for visualization design for each CWE. Thus, while the basic requirements will not change across elements, the values of parameters associated with modeling of elements will change according to the context of each requirement. Consider a requirement to compare an air attack result against a target, located close to a mosque, with the intended point of attack to assess progress toward an effect. In this case the sensitivity parameter for the comparison would be set to a high value, since collateral damage to the mosque would lower the assessment of success toward effect. The comparison of planned to actual result would indicate success only if the attack were extremely precise, that is, resulted in no damage to the mosque.

By this method we develop visualization concepts for each primitive within the context of the system requirements. Common combinations of requirement and cognitive work elements are collected together into common visualization concepts. The individual concepts then are aggregated into higher-level collections to form visualizations at the screen level. This process is iterated against the CORE system model, thereby allowing validation of visualizations by ensuring that the system follows the processes outlined in that model.

3.0 Integrating with GUI Builders

Good human factors designs are frequently "lost in translation" between the original interface designers and those ultimately responsible for system implementation, much to the dismay of all parties. This is due largely to a mix of communication and technical issues. In fact, many software and systems engineers are ignorant of human factors issues altogether, which makes the inclusion of carefully designed solutions practically impossible. One potential solution to this problem is to include the work elements concept (which is one part of a full human systems interface (HSI) solution) into the tools used by the software community.

GUI builder applications are often the centerpiece for software development. In addition to providing a visual method for designing a software system's interface, many high-quality GUI builders offer features such as round-trip software engineering, direct access to software repositories, and the inclusion of robust integrated development environments (IDEs). Some of these utilities even support multiple programming languages. Further modularity allows additional features to be added by the vendor, or even by third parties.

A practical mechanism for including CWE components is to include them within the GUI builder framework. One way to do this is to make the CWEs available as widgets within the framework, much like traditional button and menu elements are presented as widgets. A modular plug-in extension with the additional CWE widgets could be used to implement this approach.

Detailed specifications and high-level descriptions would be accessible for each CWE, either as a part of contextual popup dialogs or explicit help text. These specifications can be used to assist with the selection of relevant CWE widgets, an especially valuable feature for those implementers possessing a limited background in cognitive science. Widget tooltips and representative iconic images also will help the coder to quickly identify the specific CWE desired.

Figure 1 depicts a representative GUI builder (Glade, in this case). The area highlighted by the oval shows where a modular CWE toolbar would be loaded into the application. The intent is to make CWE as well-integrated into the application as the rest of the widget set. Therefore, the "look and feel" of the CWE module is expected to be as similar to that of the native widgets as possible, while simultaneously providing the specific capabilities of the CWE functions.

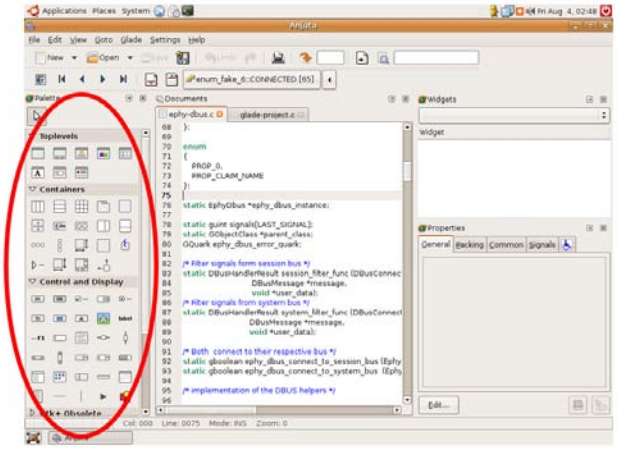

Figure 1 Enhancing the GUI builder

A programmer's selection and placement of a specific CWE into the design accomplishes several tasks. First, suggestions for implementing the CWE are presented to the programmer. (There may be several ways the CWE can be implemented or represented.) The CWE itself may be a visually-oriented or algorithmically-oriented component. For algorithms, a wizard will assist the programmer by providing a selection of relevant algorithms, their descriptions, specifications, and references. For visual components, the wizard might display the various visual representation options to the programmer.

In addition; operational code, code templates, or pseudo-coded (comment-based) solutions can be directly inserted into the code base. The options selected from the wizard indicate the CWE implementation to be included in the code. The generated code is managed within the GUI builder just as with the other drag-and-drop widgets.

Another benefit is that comments can be injected into the code surrounding the CWE implementation. These comments are explicit notations that CSE issues are directly addressed within the coded application. The construction wizard should allow the programmer to include clear text, to be added as comments corresponding to the CWE being constructed.

Finally, comments in the code provide a record of traceability back to the system's requirements. This encourages quality assurance by linking the specific requirement to the reason a particular feature or operation is implemented in a certain way. The requirement identifier is assumed to be a clear text string that can be captured by the CWE construction wizard and automatically included in the generated code.

Again, consider the CWE "compare." When the programmer has selected and placed this widget, a wizard introduces a series of questions: What requirement does this meet? Is this a visual or a conceptual comparison? Visual comparison might include options to show items side by side within a tabbed window, as a popup, or interlaced (as in visual code diffs). Should the display be hard-coded, or selectable by the end user? Perhaps the end user's requirement is only to examine the differences between several items, or perhaps the end-user must make a selection based on the comparison. If a selection is to be made, how will that selection be indicated to the system? Does the system make a recommendation to the user? (If so, what is the name of the method or function to be called to assist with that comparison?) If the "compare" CWE is conceptual only, then the wizard might ask the programmer for the method or function names to call for the comparison, and how to indicate the status of the operation. The wizard can be used to guide the coder through all of these issues, and decisions can be captured and annotated as comments in the code. This information will be valuable for justifying implementation decisions.

4.0 Conclusion

We have described an often-ignored, but important, component of software system design: The cognitive aspect. In this paper, we described a method that not only encourages the inclusion of cognitive components into the design, but also introduces a practical mechanism through which software implementers can directly incorporate key cognitive aspects into the code.

One reason it is important to define such cognitive components is to ensure that human cognitive needs and expectations are properly included at the system specification level, early in the design process. Systems engineers can use these definitions as key inputs by incorporating them into early definition processes and products (DoDAF, etc.) See [4--7] for examples of integrating cognitive work requirements into the design

process.

Another reason a mechanical approach to including cognitive components is needed is to provide a mapping from the software directly to the cognitive requirements. Such traceability is an important part of a unified systems engineering process.

5.0 References

[1] Conrad, K., & Stanard, T. (2007). Advanced design patterns: Enabling designers of complex systems. *Proceedings of the 2007 International Conference on Software Engineering Research and Practice*, 1-7.

[2] Stanard, T., Osga, G., Wampler, J., & Conrad, K. (2006). HCI Design Patterns for C2: A vision for a DoD design reference library. *Proceedings of the 2006 Command & Control Research and Technology Symposium*, San Diego, CA.

[3] Stanard, T., & Wampler, J. (2005). Work-centered HCI design patterns. *Proceedings of INTERACT 2005: Communicating Naturally through Computers*, Rome, Italy.

[4] Christopher R. Hale. Executable Requirements for Visualization Design. 2006 Human Factors and Ergonomics Society Conference. San Francisco, CA.

[5] Christopher R. Hale. Visualization Design Using an Integrated Joint Cognitive System Development Methodology. 2008 Human Factors and Ergonomics Society Conference.

[6] Christopher R. Hale and Vincent Schmidt. Four Challenges, and a Proposed Solution, for Cognitive System Engineering – System Development Integration. 2008 Industrial Engineering Research Conference. Vancouver, BC.

[7] Dave O'Malley, Jon Zall, John Colombi, and Joe Carl. Integrating Cognition into System Design. 2008 International Conference on Software Engineering, Research, and Practice. Las Vegas, NV.